MUSIQUE : Descriptions de procédures de lecture, Procédures de description de lectures, Procédures de lecture de descriptions.

Pat GREUSSAY

INTRODUCTION

Il est reconnu que les programmes de composition musicale produisent de la musique insignifiante. Cette carence est due, c'est évident, à la sous-estimation à peu près générale de la complexité des descriptions nécessaires à spécifier des musiques élaborées.

Je crois que cette carence n'est pas moins due à la sous-estimation des possibilités descriptives offertes par un certain nombre d'objets conceptuels utilisés en programmation des ordinateurs.

Il est clair qu'un programme de composition devrait comporter une composante analytique très puissante, mais je crois que nous savons encore trop peu de choses sur la façon de décrire précisément nos analyses. Je crois également qu'il est trop tôt pour construire de telles composantes.

Il me semble plus nécessaire à l'heure actuelle d'essayer d'exprimer ce que nous savons, et les musiciens sont très savants, d'une façon suffisamment précise et explicite, pour que ce savoir puisse être mis sous forme de programmes.

Il s'agit donc de mettre en machine des *descriptions*.

Nous pouvons lire des oeuvres musicales. Nous savons également qu'une oeuvre est un objet temporel. Le temps intervient dans l'exécution, la lecture, l'analyse et la composition de l'oeuvre.

Sauf à écrire de la musique ou des programmes, nous ne savons pas très bien comment décrire des processus temporels et leurs interactions.

C'est en ceci que la programmation est intéressante, pour les concepts qu'elle apporte et met en jeu dans la description de processus temporels.

Je vais essayer de prendre au pied de la lettre la comparaison souvent faite entre, d'une part le texte d'un programme et ce programme en activité, d'autre part la partition d'une oeuvre et cette oeuvre en exécution ou en lecture.
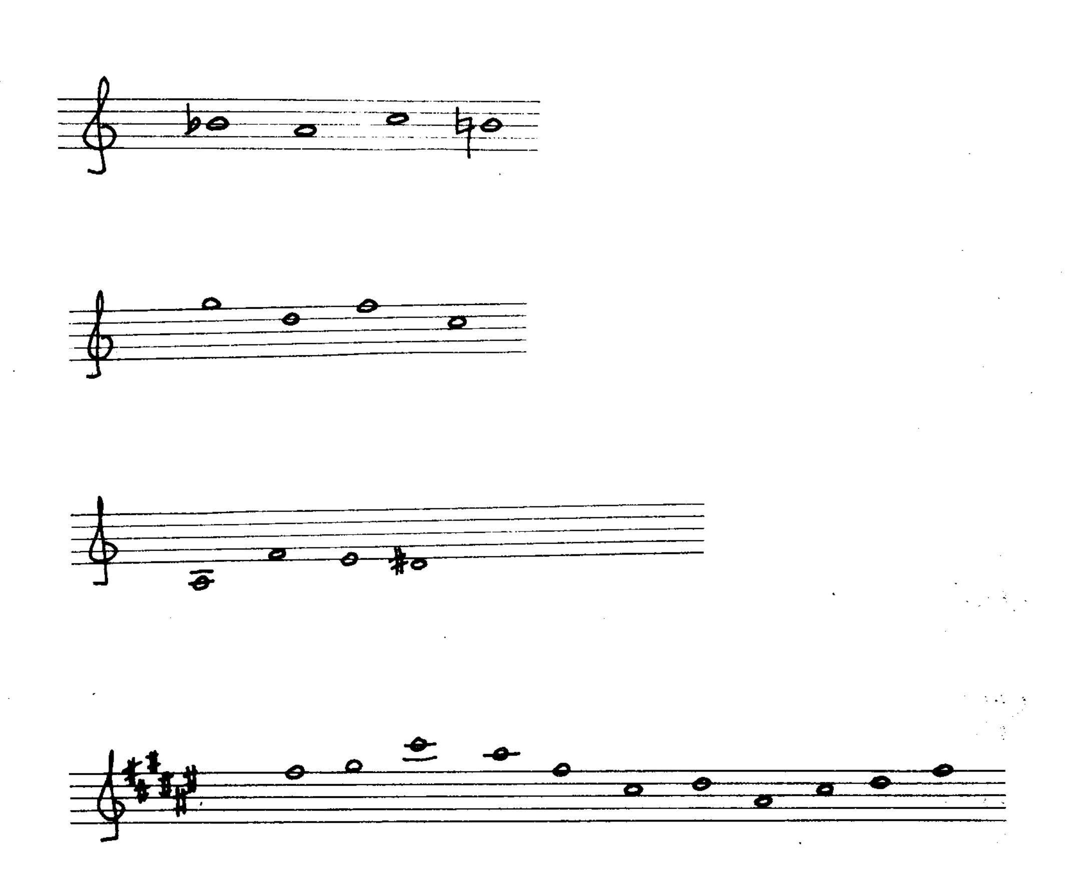
Une des raisons avancées pour rendre compte de la médiocrité des programmes d'analyse ou de composition, c'est qu'ils sont sans mémoire. Mais comment organiser la mémoire? Et encore quoi mettre en mémoire? Au préalable à l'analyse il y a un m oment de lecture, probablement déterminant. Cette lecture suppose un savoir pré-existant. Je ne m'occupe pas de la manière dont ce savoir a été acquis. Plutôt je cherche ici à savoir comment le décrire, et à savoir comment ces descriptions peuvent agir.

Je suppose que ce savoir est par lui-même *effectif*.

Dans ces essais, je travaille sur de la musique tonale, supposée être composée. Je n'y tiens pas particulièrement. Reste que c'est commode : il y a une structure logique explicitable en partie, et je pense que les différents sur les notions de bases y seront les moindres.

I

Je *peux* lire de la musique. Cette lecture prend du temps.

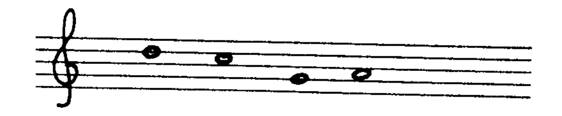


Lire un de ces fragments me met dans un état d'attention d'un type très spécial (très spécialisé), sans pour autant qu'un processus d'analyse ait été proprement lancé.

Un musicien possède probablement un grand nombre de schèmes de lecture qu'il n'applique pas encore à ce stade, mais q'il *appelle*, qu'il rend disponible à une application éventuelle.

Je vais essayer de préciser ces idées en décrivant des morceaux de programmes. Ces programmes sont écrits dans le langage CONNIVER.

Je lis ce fragment :



Supposons que je cherche à le spécifier sous forme de relations de *dominante* entre les notes prises par couple.

```
(ADD '(D SOL RE))

(ADD '(D DO SOL))

(ADD '(D RE LA))
```

Si j'ai un instant d'oubli, ou que je me demande ce que j'ai lu, je peux poser des questions :

```
(PRESENT '(D SOL RE))
 (D SOL RE)
```

Ce fragment comporte-t-il une relation de D entre sol et ré?

```
(PRESENT '(D RE !X))
 (D RE LA)
```

Ce fragment comporte-t-il la D de ré?

```
(PRESENT '(!X !Y SOL))
 (D DO SOL)
```

Y-a-t-il une note dans ce fragment dont sol est la D?

Questions partielles. Je peux souhaiter obtenir toutes les relations connues jusqu'ici. Il s'agit d'une attitude différente, je veux savoir *toutes* les relations d'un type donné. Je définis une fonction à cet effet et je l'applique :

```
(DE EVERY (SQ)
          (EVAL (LIST 'FOR-EACH SQ '(PRIN1 CURRENT)))
          (TERPRI))

(EVERY '(D !X !Y))
 (D RE LA) (D DO SOL) (D SOL RE)
```

A présent, puis-je, à partir de ce que je sais, décrire des relations dérivées? par exemple de sous-dominante :

```
(PRESENT '(SD RE SOL))
 NIL
```

Evidemment pas, si je ne sais pas ce qu'est une SD. Que sais-je?

$\forall$ x,y   D(y,x) $\supset$ SD(x,y)

Etre en relation de SD, c'est quelquechose que je peux *déduire*, j'en fais donc une méthode

```
(IF-NEEDED IS-SD (SD ?X ?Y)
      %AUX% (X Y)
      (FOR-EACH (D $Y $X) (NOTE)))
```
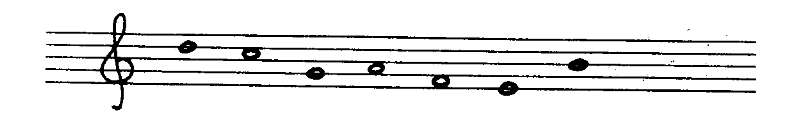
Et je pose une question :

```
(PRESENT '(SD RE SOL))
 (SD RE SOL)
```

Puis toutes les questions concernant cette nouvelle relation

```
(EVERY '(SD ?X ?Y))
 (SD LA RE) (SD SOL DO) (SD RE SOL)
```

Je poursuis ma lecture



```
(ADD '(D LA MI))

(ADD '(D FA DO))

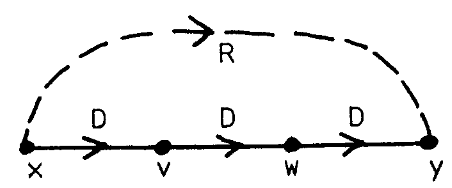
(ADD '(D MI SI))
```

Je m'interesse maintenant à un autre type de relation dérivée, moins simple : la notion de relative. Et je demande :

```
(PRESENT '(R ?X ?Y))
 NIL
```

Il faut *décrire* ce que je sais de cette relation. Essayons une description:


$$\forall x,y \quad \exists v,w \quad D(x,v) \wedge D(v,w) \wedge D(w,y) \supset R(x,y)$$




```
(IF-NEEDED IS-R (R ?X ?Y)
      %AUX% (X Y V W (YVAL Y))
      (FOR-EACH (D $X !V)
                (AND (PRESENT '(D ,V !W))
                     (PRESENT '(D ,W $Y))
                     (NOTE)
                     (SETQ Y YVAL))))
```

et je me demande si je vois une occurence de relatives dans ce fragment :

```
(PRESENT '(R ?X ?Y))
 (R FA RE)
```

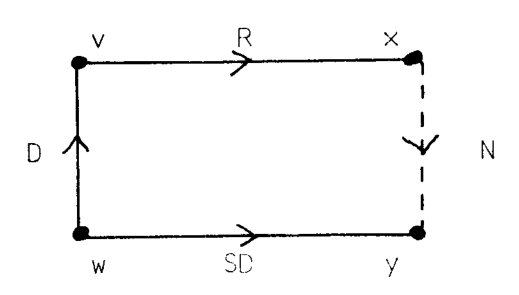
Puis toutes les relatives :

```
'(EVERY '(R ?X ?Y))
 (R FA RE) (R RE SI) (R DO LA) (R SOL MI)
```

Je pourrais également chercher les napolitaines. Mais je vais changer légèrement de point de vue. Au fond j'aurai pu essayer de les repérer à la volée. Au fur et à mesure de la lecture. Et pas en réfléchissant et en me posant des questions. C'est une façon d'exprimer cet état où une relation est *préoccupante*, où elle saute aux yeux, où on la *guette* à tout moment.

J'en fais alors un démon qui, *à la lecture* même du fragment, sans que je me pose de questions, essaye de repérer les N s'il y en a.

$$\forall x,y \quad \exists w,v \qquad D(w,v) \wedge R(v,x) \wedge SD(w,y) \supset N(x,y)$$




```
(IF-ADDED GUET-N (D !X !Y)
      %AUX% (X Y V W)
      (ASSUMING (SD ,Y ,X)
                (FOR-EACH (D !W !V)
                          (AND (PRESENT '(R ,V ?X))
                               (PRESENT '(SD ,W ?Y))
                               (TTAB 25)
                               (PRINT =(JE VOIS (N ,X ,Y))))))))
```

Et je reprend au début ma lecture du fragment.



```
(ADD '(D SOL RE))
(ADD '(D DO SOL))
(ADD '(D RE LA))
(ADD '(D LA MI))
(ADD '(D FA DO))
                        (JE VOIS (N MI FA))
(ADD '(D MI SI))
                        (JE VOIS (N MI FA))
```

La lecture (le programme) a manqué la relation

(N SI DO)

En effet, pour la repérer il aurait fallu

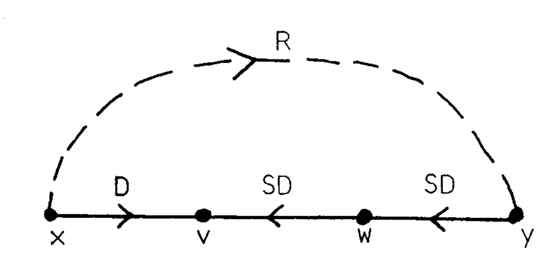
(D SOL RE) qui existe bien

et (R RE SI)

Mais pour que le démon reconnaisse la présence de la relation (R RE SI), il lui aurait fallu la relation (D MI SI) qu'il ne connait pas encore : GUET-N, activé par (D MI SI) intervient, mais *suppose* (par le ASSUMING) la relation de SD (ce qui semble naturel pour une napolitaine), i.e. SI - MI.

En conservant le même démon pour les napolitaines, je dois donc modifier ma définition de *relative* : le repérage se faisant par les sous-dominantes :

$$\forall x,y \quad \exists v,w \qquad D(x,v) \land SD(w,v) \land SD(y,w) \supset R(x,y)$$




```
(IF-NEEDED IS-R (R ?X ?Y)
      %AUX% (X Y V W (YVAL Y))
      (FOR-EACH (D $X !V)
                (AND (PRESENT '(SD !W ,V))
                     (PRESENT '(SD $Y ,W))
                     (NOTE)
                     (SETQ Y YVAL))))
```

Et je reprend une nouvelle fois ma lecture au début :

```
(ADD '(D SOL RE))

(ADD '(D DO SOL))

(ADD '(D RE LA))

(ADD '(D LA MI))

(ADD '(D FA DO))
                              (JE VOIS (N MI FA))

(ADD '(D MI SI))
                              (JE VOIS (N MI FA))
                              (JE VOIS (N SI DO))
```

II

Je vais maintenant changer de point de vue. Je peux supposer que ce savoir harmonique élémentaire n'est pas déductif, mais "procédurel", i.e. je peux construire des *spécialistes* très bêtes mais très efficients pour répondre à des questions limitées :

(GIV relation note) me donne la note qui est dans la relation spécifiée avec la note-argument.

(IS relation note 1 note 2) sait reconnaitre si la relation spécifiée tient entre note 1 et note 2.

```
(DE GIV (RELATION NOTE)
        (CADR (MEMQ NOTE (GET ALLREL RELATION))))

(DE IS (RELATION NOTE1 NOTE2)
       (EQ NOTE2 (GIV RELATION NOTE1)))

(RPLACA 'ALLREL '(
      N (LA+ SI DO RE-                N-EXIST
         SOL+ LA SI- DO-              N-EXIST
         RE+ MI FA SOL-               N-EXIST
         SI+ DO+ RE MI- FA-           N-EXIST
         MI+ FA+ SOL LA-              N-EXIST)
      S (RE- DO SI LA+                N-EXIST
         DO- SI- LA SOL+              N-EXIST
         SOL- FA MI RE+               N-EXIST
         FA- MI- RE DO+ SI+           N-EXIST
         LA- SOL FA+ MI+              N-EXIST)
      D (FA- DO- SOL- RE- LA-
         MI- SI- FA DO SOL
         RE LA MI SI FA+
         DO+ SOL+ RE+ LA+ MI+ SI+     N-EXIST)
      R (SI+ RE+ FA+ LA DO MI- SOL-  N-EXIST
         LA+ DO+ MI SOL SI- RE- FA-  N-EXIST
         MI+ SOL+ SI RE FA LA- DO-   N-EXIST)
            ))
```

Je noterai :

OB+ =df ♯ OB

OB- =df ♭ OB

En pratique je n'utiliserai pas dans ce qui suit les doubles ♯ ou ♭, c'est la raison de la valeur par défaut N-EXIST.
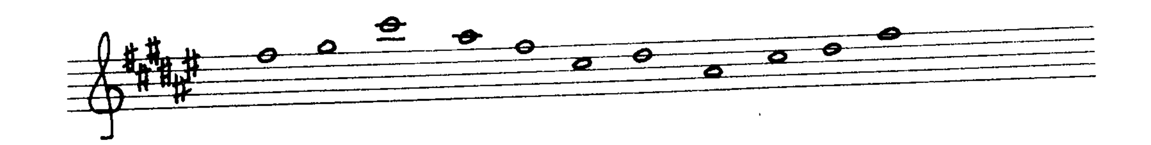
Cette idée de repérer des relations *à la volée* semble féconde. Je puis également inclure dans ce repérage une trace du moment où il a eu lieu, ou plus simplement garder une trace de l'ordre dans lequel le repérage s'est effectué.
Je spécifie à cet effet une macro qui, invoquée, me produit un objet unique, différent de tous ceux qui l'ont précédé :

```
(MACRO : (LAMBDA () '@(SETQ 20N (ADD1 20N))))
(SETQ 20N 0)
```

Puis je définis un nouveau démon qui, à chaque fois qu'une note est lue, cherche s'il peut dégager des relations de dominante avec les notes précédemment lues.

```
(IF-ADDED N-O-T (NOT !X)
      %AUX% (X Y)
      (AND (ABSENT '(NOT ,X))
           (FOR-EACH (NOT !Y)
                     (COND
                       ((IS 'D X Y) (ADD '(: D ,X ,Y)))
                       ((IS 'D Y X) (ADD '(: D ,Y ,X)))
                          ))))
```

Je lis alors :



```
(MAPC '(FA+ SOL+ DO+ LA+ FA+ DO+ RE+ LA+ DO+ RE+ FA+)
      '(LAMBDA (X) (ADD '(NOT ,X))))
```

A ce point, je demande quelles sont les relations de dominante repérées. Les nombres inclus dans les items me permettent de savoir dans quel ordre a eu lieu ce repérage.

```
(EVERY '(IN D !X !Y))
 (4 D SOL+ RE+) (3 D RE+ LA+) (2 D FA+ DO+) (1 D DO+ SOL+)
```

Pour certaines raisons, je peux être amené à me demander si ces dominantes peuvent se mettre en chaîne en formant des fragments de cycle de quintes.

Je définis alors un nouveau spécialiste pour répondre à cette question.

```
(DE CYCLE-5 (;; N1 N2 X Y Z NX WCTX)
        (FOR-EACH (!N1 D !X !Y)
             (AND (PRESENT '(!N2 D ,Y !Z))
                  (ADD '(CPL ,N1 ,N2) 'WCTX)))
        (WHILE (PRESENT '(CPL !N1 !N2) 'WCTX)
               (WHILE (PRESENT '(CPL !NX ,N1) 'WCTX)
                      (SETQ N1 NX))
               (PRESENT '(,N1 D !X !Y))
               (PRIN1 X)
               (PRIN1 Y)
               (WHILE (PRESENT '(CPL ,N1 !N2) 'WCTX)
                      (REMOVE '(CPL ,N1 ,N2) 'WCTX)
                      (PRESENT '(,N2 D !X !Y))
                      (PRIN1 Y)
                      (SETQ N1 N2))
               (PRIN1 '/))
        (TERPRI))
```

Voici comment l'objet fonctionne.

Si j'entre par exemple :

(NOT DO)

(NOT MI)

(NOT FA+)

(NOT RE)

(NOT SI)

(NOT SOL)

(NOT FA)

la méthode N-O-T crée les items

(1 D SI FA+) (2 D MI SI) (3 D SOL RE)

(4 D DO SOL) (5 D FA DO)

chacun des items créés est de la forme

(n D note1 note2)

Le spécialiste CYCLE-5 crée un contexte de travail WCTX. Dans ce contexte spécialisé, il va ranger des items qu'il crée, de la forme : (CPL n1 n2) ou n1 et n2 sont les n^os de dominante, tels que note2 de n1 est identique à note1 de n2.

Dans notre exemple, on aura dans WCTX

(CPL 2 1)

(CPL 5 4)

(CPL 4 3)

Puis il recherche les premiers éléments de chaque chaîne de CPL, imprime la chaîne, élimine la chaîne, et recommence tant qu'il reste des CPL dans WCTX. Ce qui donnera dans notre exemple :

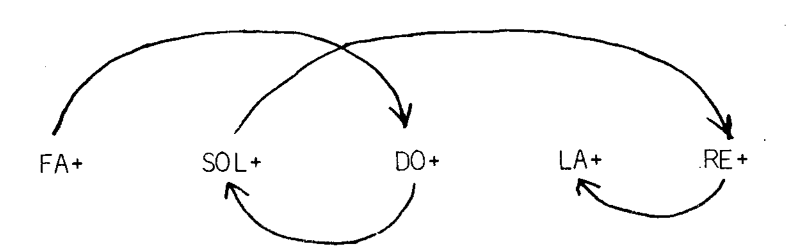
```
(CYCLE-5)
FA DO SOL RE / MI SI FA+ /
```

Si j'applique l'objet au fragment de Debussy, j'obtiens :
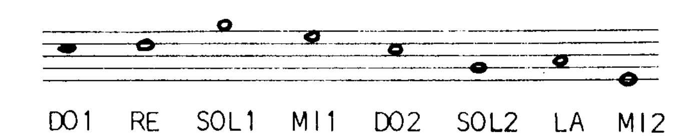
```
(CYCLE-5)
FA+ DO+ SOL+ RE+ LA+ /
```

à partir des objets dans le contexte :

```
(*ITEM (8 D SOL+ RE+))
(*ITEM (7 D RE+ LA+))
(*ITEM (6 D FA+ DO+))
(*ITEM (5 D DO+ SOL+))
(*ITEM (NOT RE+))
(*ITEM (NOT LA+))
(*ITEM (NOT DO+))
(*ITEM (NOT SOL+))
(*ITEM (NOT FA+))
(*METHOD IF-ADDED N-O-T)
```



Changeons encore une fois légèrement de point de vue.
Je suppose que ma lecture a affaire à un ensemble de notes, ensemble ordonné par la relation APRES = A



et constitué dans le contexte comme par exemple

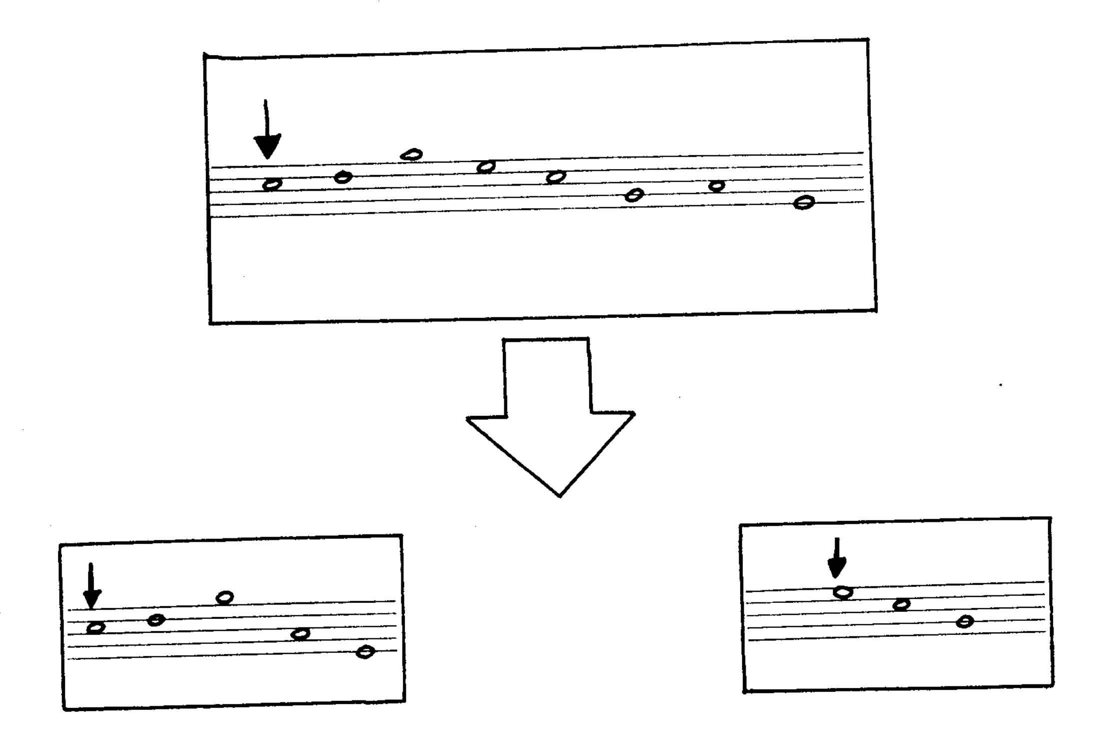
(A SOL2 DO2)
(A SOL1 RE)
(A LA SOL2)
(A RE DO1)
(A MI2 LA)
(A MI1 SOL1)
(P DO1)
(A DO2 MI)

DO1 est *premier* de la chaîne, ou segment.
Je peux construire un spécialiste capable de me livrer tous les segments présents dans le contexte.

```
(DE SEGMENTS (;; X Y)
                (FOR-EACH (P !X)
                          (PRIN1 X)
                          (WHILE (PRESENT '(A !Y ,X))
                                 (SETQ X (PRIN1 Y)))
                          (TERPRI)))
```

```
(SEGMENTS)
DO1 RE SOL1 MI1 DO2 SOL2 LA MI2
```

Pour certaines raisons, je peux envisager de segmenter le fragment lu, par exemple



obtenant ainsi le nouveau contexte

(P DO1)	(P MI1)
(A RE DO1)	(A DO2 MI1)
(A SOL1 RE)	(A SOL2 DO2)
(A LA SOL1)	
(A MI2 LA)	

Pour conserver la cohérence de mes objets ordonnés, je devrai prendre certaines précautions. Ces précautions seront décrites par un ensemble de démons spécialistes. Ces démons s'activeront sur présentation ou élimination de l'item.

(A note1 note2)

```
(IF-ADDED -1ER (A !X !Y)
      %AUX% (X)
      (AND (PRESENT '(P ,X))
           (REMOVE '(P ,X))))
```

Si X était premier, il ne doit plus l'être à présent.

```
(IF-ADDED X-SUC-QQ1 (A !X !Y)
      %AUX% (X Z)
      (AND (PRESENT '(A ,X !Z))
           (REMOVE '(A ,X ,Z))))
```

Si X était le successeur d'un Z, il ne doit plus l'être.

```
(IF-ADDED QQ1-SUC-Y (A !X !Y)
      %AUX% (Y Z)
      (AND (PRESENT '(A !Z ,Y))
           (REMOVE '(A ,Z ,Y))))
```

Si Y avait un successeur Z, ce ne doit plus être le cas

```
(IF-REMOVED +1ER (A !X !Y)
      %AUX% (X)
      (ADD '(P ,X)))
```

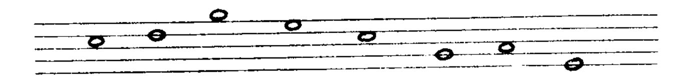
Si X n'est plus le successeur de Y, X doit devenir premier.

Au moins conceptuellement ces démons peuvent être activés en même temps.

(A note1 note2) active { -1ER, X-SUC-QQ1, QQ1-SUC-Y }

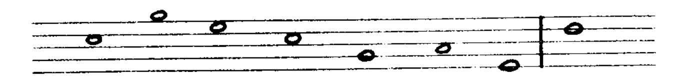
et { -1ER, X-SUC-QQ1, QQ1-SUC-Y } activent +1ER

```
(SEGMENTS)
DO1 RE SOL1 MI1 DO2 SOL2 LA MI2
```



```
(ADD '(A SOL1 DO1))

(SEGMENTS)
RE
DO1 SOL1 MI1 DO2 SOL2 LA MI2
```
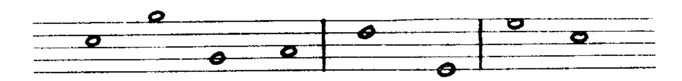


```
(ADD '(A MI2 RE))

(SEGMENTS)
RE MI2
DO1 SOL1 MI1 DO2 SOL2 LA
```
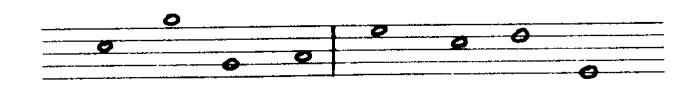


```
(ADD '(A SOL2 SOL1))

(SEGMENTS)
MI1 DO2
RE MI2
DO1 SOL1 SOL2 LA
```



```
(ADD '(A RE DO2))

(SEGMENTS)
MI1 DO2 RE MI2
DO1 SOL1 SOL2 LA
```
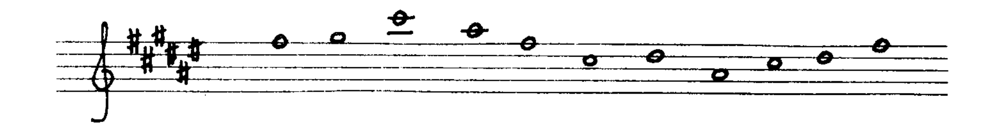


Il est clair que je ne dois pas essayer de former ici une chaîne circulaire, la chaîne perdant alors son premier ne serait plus repérable par le spécialiste SEGMENTS.

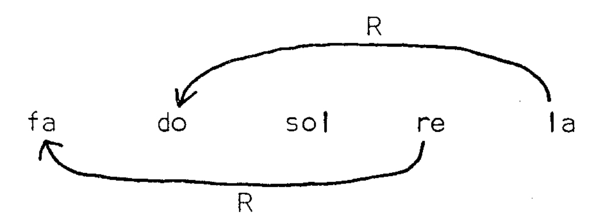
III

Les lectures envisagées jusqu'ici étaient toutes des lectures "à l'instant" ou bien "à l'instant et en arrière". Je m'interesse ici à des lectures "en avant". Dans certains cas, nous sommes dans des situations où nous avons des raisons de penser qu'un objet bien spécifié devrait arriver, quitte à être déçus s'il n'arrive pas, ou encore à oublier simplement qu'on l'avait prévu. Je voudrai bien exprimer précisèment le fait que parfois, je poste des guetteurs dans l'avenir de ma lecture.

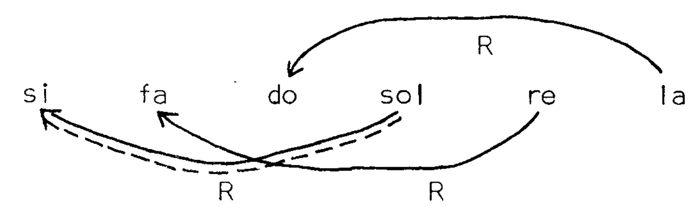
Voici un exemple d'une telle situation.



J'y recherche les relatives :



et je constate que sol n'a pas de relative (et n'arrive qu'une fois). J'attend ou je guette le si dans ce qui suit



Ce moment de la lecture, je pourrai l'exprimer ainsi

repérage des relatives

```
(FOR-EACH (NOT !X)
          (CEXIST (NOT !Y)
                  (COND
                    ((IS 'R Y X)
                     (ADD '(R ,Y ,X)))
                    ((FAIL)))))
```

```
(EVERY '(R !X !Y))
 (R LA+ DO+) (R MI+ SOL+)
```

Puis, pour chacune des notes X présentes, si elle n'est pas inscrite dans une relation de relative, je poste un guetteur qui attend son arrivée dans le futur de la lecture. Une sorte de chat endormi, à la moustache sensible à la note qui est la relative de X.

```
(FOR-EACH (NOT !X)
      (OR (PRESENT '(R !Y ,X))
          (PRESENT '(R ,X !Y))
          (GUETTER (GIV 'R X))))
```

GUETTER est une fonction assez complexe qui poste le guetteur.

```
(DE GUETTER (X Y Z)
            (PRINT =(JE GUETTE ,X))
            (PUT X Z 'RAISON)
            (ADD (EVAL
                    =(IF-ADDED ,X (NOT ,X)
                    %AUX% ()
                    (PRINT @='(ENFIN ,X ARRIVE))
                    ,Y
                    (REMOVE ',X)
                    (RPLACD ',X) ))))
```

A son appel, guetter aura un argument obligatoire X, la note dont on guettera l'occurence. L'argument Z facultatif sera mis dans la p-liste de X sous l'indicateur RAISON, on pourra y placer la raison pour laquelle la note X était guettée.
Puis GUETTER *construit* un démon de même nom que X, l'inclut dans le contexte, où il attend en silence l'arrivée de l'item (NOT note-guettée). Si la note arrive, le démon se réveille, indique qu'il est en activation par la note d'éveil, accomplit l'action Y spécifiée en argument facultatif de GUETTER, puis ceci fait, il s'élimine, ayant joué son rôle, du contexte courant, et se détruit physiquement.

Dans notre exemple, le démon synthétisé sera, par l'appel de
(GUETTER (GIV 'R X)) avec X = SOL+

```
(IF-ADDED SI (NOT SI)
      %AUX% NIL
      (PRINT (QUOTE (ENFIN SI ARRIVE)))
      NIL
      (REMOVE (QUOTE SI))
      (RPLACD (QUOTE SI)))
```

qui attend l'arrivée de la note (NOT SI). Si elle arrive (et elle arrive en effet), le démon signalera
"ENFIN SI ARRIVE" puis se détruira.

Cependant, quand le démon SI a été posté, je pensais spécialement aux relatives. Or ce guetteur peut être activé alors que je n'y pense plus du tout. Je voudrai donc garder une trace de l'état d'esprit dans lequel j'étais lorsque j'ai ordonné la constitution du guetteur par

```
(GUETTER (GIV 'R X)
        =(ADD '(R ,X @(GIV 'R X))))
```

ce qui me donnera le démon

```
(IF-ADDED SI (NOT SI)
    %AUX% NIL
    (PRINT '(ENFIN SI ARRIVE))
    (ADD '(R SOL+ SI))
    (REMOVE 'SI)
    (RPLACD 'SI))
```
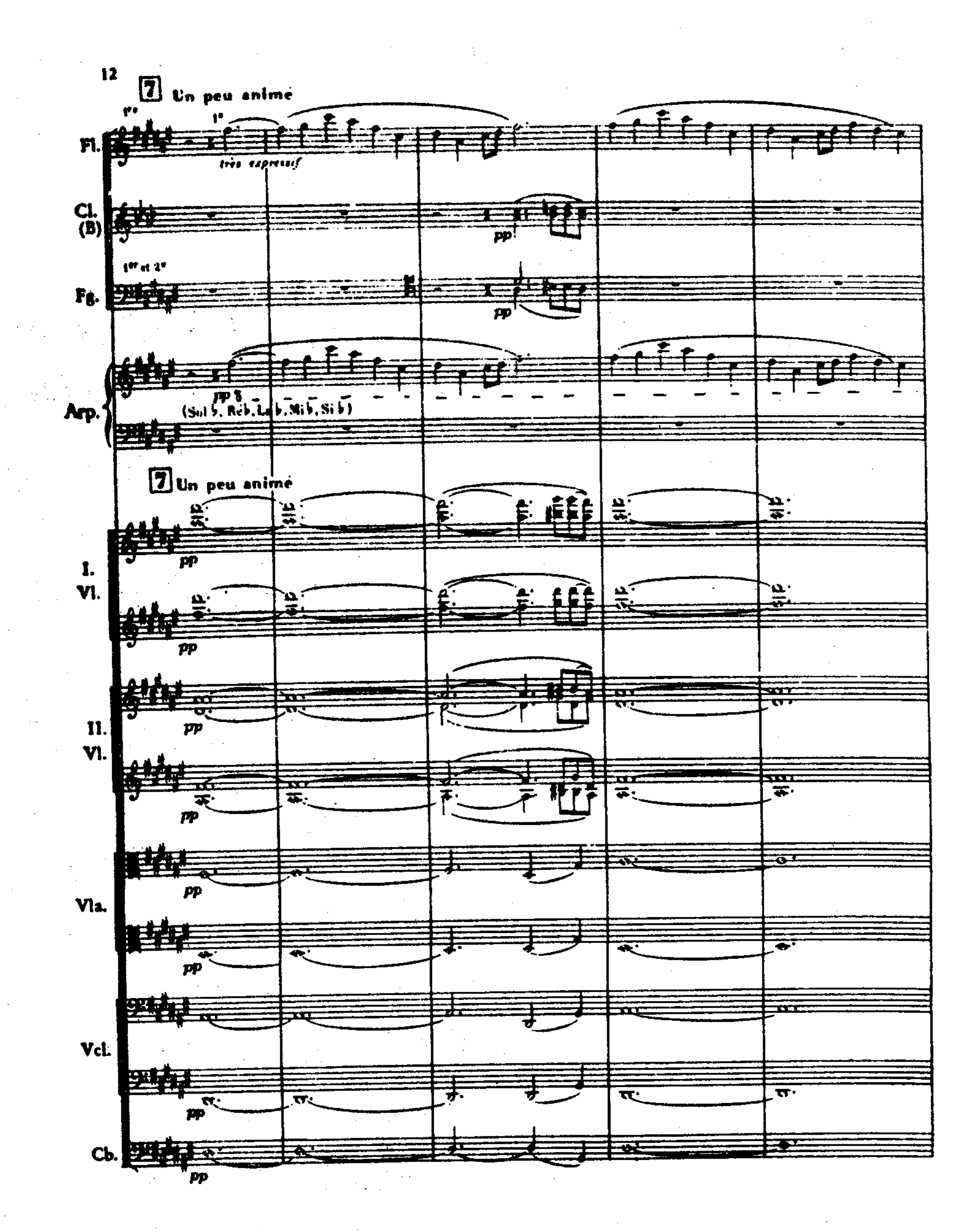
qui, lorsque SI arrivera, ajoutera au contexte l'item

```
    (R SOL+ SI)
```

Donc, GUETTER me permet également de lancer des actions à retardement.

Voyons d'autres raisons de mettre en place des guetteurs. Toujours avec le même exemple.
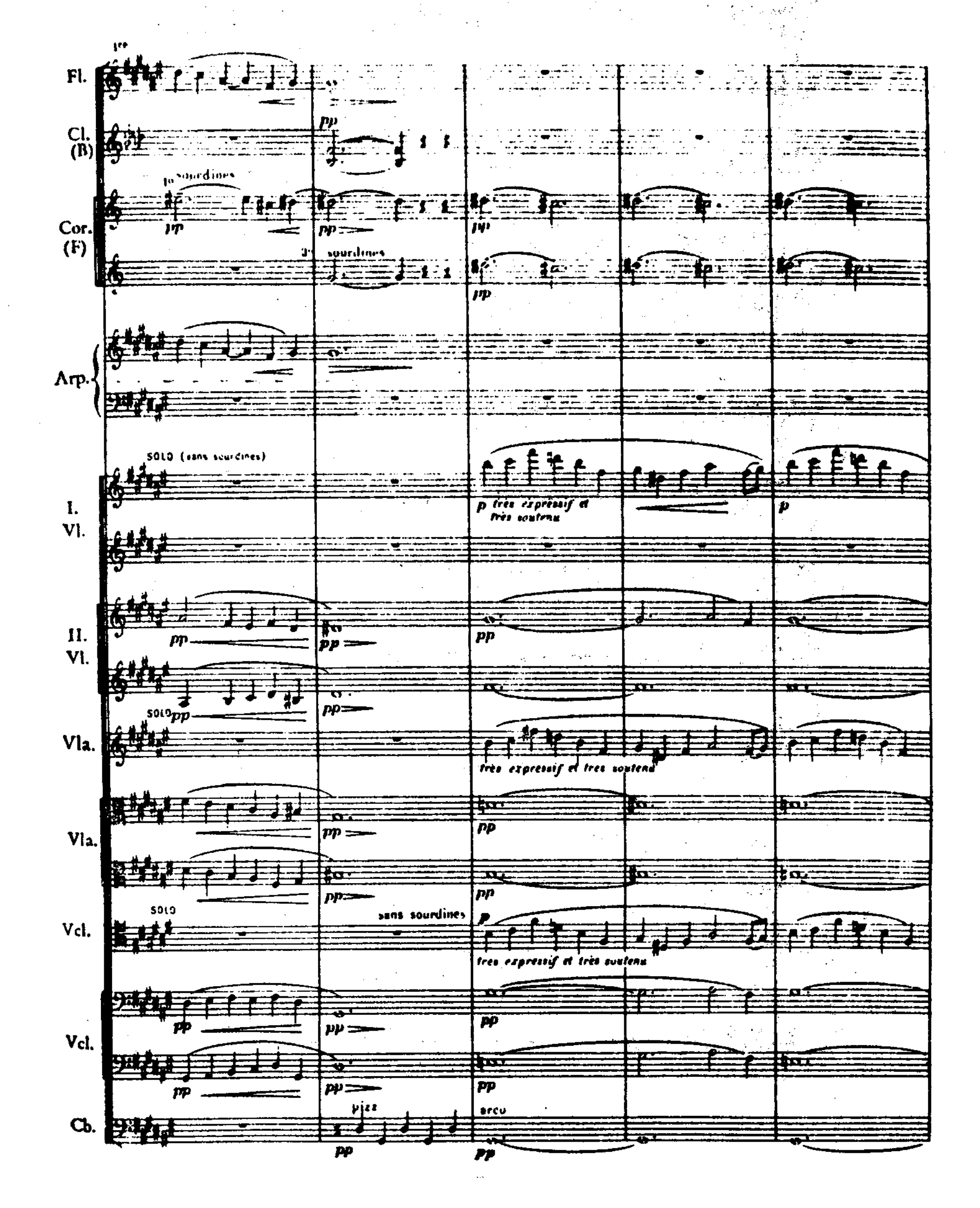
C'est le n° [7] du premier Nocturne de Debussy.

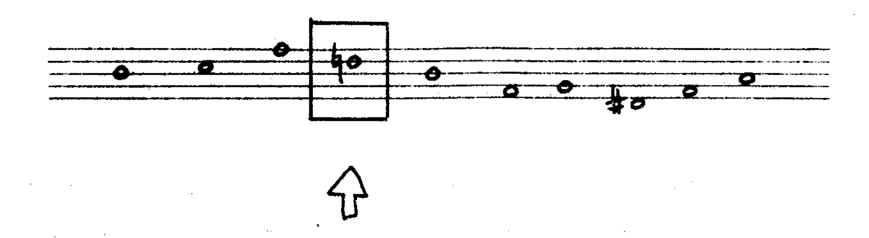


L'arrivée du SI ♯ , sensible de DO ♯ , pourrait me faire utiliser une règle de lecture que voici :

> "S'il existe une sensible de X, guetter la napolitaine de X.
> S'il existe une napolitaine de X, attendre la sensible de X."

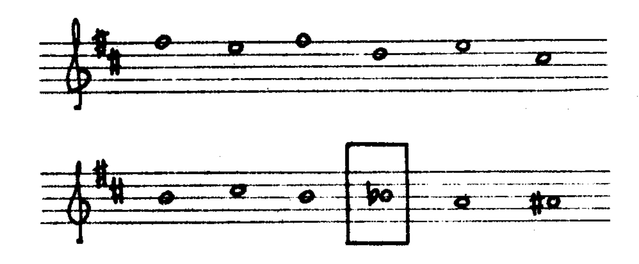
Or j'ai repéré SI♯ sensible de DO♯. Je ne suis pas trop surpris quand arrive, quelques mesures plus tard





i.e. le RE♮ napolitaine de DO♯.

Restons dans Nuages, à la mesure 1 cette fois.



L'arrivée du SI♭ , napolitaine de LA me suggère de mettre en jeu le démon que voici :

```
(IF-ADDED N-A-P (N !X !Y)
      %AUX% (X Y Z POLE1 POLE2 RELATION)
      (SETQ Z (GIV 'S X))
      (GUETTER Z
               =(PRINT (GET ',Z 'RAISON))
               =(,X AYANT SA NAP JE GUETTE SA SENS))
      (CEXIST (POLE !POLE1)
              (OR (PRESENT '(!RELATION ,POLE1 ,X))
                  (FAIL))
              (FOR-EACH (POLE !POLE2)
                 (COND
                   ((NEQ POLE1 POLE2)
                    (SETQ Z (GIV RELATION POLE2))
                    (GUETTER (GIV 'N Z))
                    (GUETTER (GIV 'S Z))))))))
```

L'idée c'est : si j'ajoute au contexte la relation

(N x y)

. guetter la sensible de x, i.e. créer un démon qui, s'il repère la sensible de x indique qu'il la guettait parce que x avait sa napolitaine.

. s'il existe (CEXIST) un pôle POLE1 avec une RELATION avec x, pour tous les autres pôles POLE2 , guetter la sensible et la napolitaine de la note qui a avec POLE2 la même relation que x a avec POLE1.

Dans notre exemple, avec le contexte

(POLE SI)

(POLE RE)

(D RE LA)

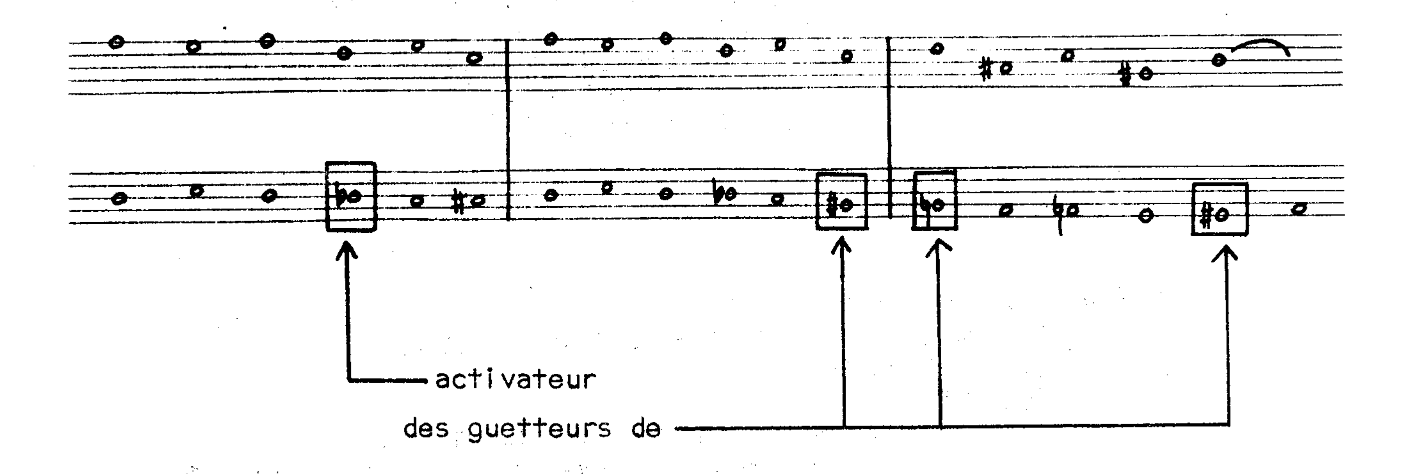
j'obtiendrai, si j'ajoute au contexte la relation

```
(ADD '(N LA SI-))
 (JE GUETTE SOL+)
 (JE GUETTE SOL)
 (JE GUETTE MI+)
```

En effet l'activateur est (N LA SI-), or on a (D RE LA) et je sais que (POLE RE). Par ailleurs le démon N-A-P sait que (POLE SI), que la D de SI est FA#, et que la N de FA# est SOL et que la S de FA# est MI# .

Il met en place, par conséquent des guetteurs qui attendent SOL et MI#.

Et le démon N-A-P n'a pas tort :



et j'obtiendrai donc bientôt

```
(ADD '(NOT SOL+))
 (ENFIN SOL+ ARRIVE)
 (LA AYANT SA NAP JE GUETTE SA SENS)

(ADD '(NOT SOL))
 (ENFIN SOL ARRIVE)

(ADD '(NOT MI+))
 (ENFIN MI+ ARRIVE)
```

Lorsque le guetteur posté, par exemple (NOT SOL) se réveille et me signale

(ENFIN SOL ARRIVE)

je puis souhaiter, ayant complètement oublié les raisons pour lesquelles je voulais que le SOL soit guetté, lui demander pourquoi le fait que le SOL arrive est interessant.

J'utiliserai à cet effet le spécialiste

```
(DE LIRE-RAISONS (NOTE ;; L)
        (ESCAPE EXIT
                (SETQ L (GET NOTE 'RAISON))
                (WHILE L
                       (OR (READ) (EXIT))
                       (PRINT (NEXTL L)))
                (PRINT 'VOILA)))
```

et je modifierai ainsi le démon N-A-P

```
(IF-ADDED N-A-P (N !X !Y)
      %AUX% (X Y Z POLE1 POLE2 RELATION)
      (SETQ Z (GIV 'S X))
      (GUETTER Z
               =(PRINT (GET ',Z 'RAISON))
               =(,X AYANT SA NAP JE GUETTE SA SENS))
      (CEXIST (POLE !POLE1)
              (OR (PRESENT '(!RELATION ,POLE1 ,X))
                  (FAIL))
              (FOR-EACH (POLE !POLE2)
                 (COND
                   ((NEQ POLE1 POLE2)
                    (SETQ Z (GIV RELATION POLE2))
                    (GUETTER (GIV 'N Z)
                             =(LIRE-RAISONS '@(GIV 'N Z))
                             =((@(GIV 'N Z) NAP DE ,Z)
                               (,Z ,RELATION DE ,POLE2)
                               (,POLE2 POLE)
                               (,POLE1 POLE)
                               (,X ,RELATION DE ,POLE1)
                               (,Y NAP DE ,X)))
                   (GUETTER (GIV 'S Z)))))))
```

Je (ou un démon, ou une autre partie d'un programme) peux alors demander pourquoi le guetteur SOL se fait ainsi remarquer. Dans un sens limité le guetteur *connaît* les raisons (il peut y avoir accès) pour lesquelles il a été posté.

Je note par "⟶" les questions posées au guetteur
et par "⟵" ses réponses.

(ENFIN SOL ARRIVE)

→ POURQUOI
← (SOL NAP DE FA+)

→ ET-ALORS
← (FA+ D DE SI)

→ ET-ALORS
← (SI POLE)

→ ET-ALORS
← (RE POLE)

→ ET-ALORS
← (LA D DE RE)

→ ET-PUIS-APRES
← (SI- NAP DE LA)

← VOILA

et après avoir livré ses raisons, le guetteur se retire.

CONCLUSION

Lire de la musique est une activité. Nous avons, face à un texte, un certain nombre de notions, certaines assez précises, d'autres tout à fait floues. Nous avons aussi certains schèmes d'explication plutôt rigides, mais également des schèmes de schèmes, des sortes de matrices qui nous permettent quelquefois, devant l'échec de nos procédures d'explication, d'en former de nouvelles, peut être plus appropriées au texte musical particulier dont il s'agit. Je crois que nous disposons à présent des outils nécessaires pour décrire ces notions, sans trop les réduire, au moyen d'ordinateurs. L'ordinateur est ici le moyen d'en attester l'effectivité. Toutefois je ne pense pas que les procédures *construites* dynamiquement, paramètrées par le texte au cours de sa lecture, soient réellement de *nouvelles* procédures. Je pense plutôt qu'il s'agit de *corrections*, d'une mise au point continue de procédures défaillantes face à un nouveau cas particulier.